COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members	Warisha Noushad (noushadw) & Zara Shahid (shahiz8)
Team Members Evaluated	Francesca Buckley (buckleyf) & Stephanie Hughes (hughes20)

Provide your genuine and engineeringly verifiable feedback. Ungrounded claims will lead to deductions. Completing the peer code evaluation on time will earn your team a total of **25 marks**. Do not exceed 2 paragraphs per question.

Peer Code Review: OOD Quality

1. **[3 marks]** Examine the main logic in the main program loop. Can you easily interpret how the objects interact with each other in the program logic through the code? Comment on what you have observed, both positive and negative features.

   To examine their main logic, I checked their Project.cpp file to understand and interpret how the objects interact with each other. Their code was easy to understand, and I could clearly observe how the objects interacted with each other because of the appropriately named variables and some comments placed throughout the Project.cpp file. Some positive features I observed was their comments present throughout the main logic code and also the appropriately named variables that helped identify what their purpose to the code was. Their main project code was honestly very well done, the only thing I could point out is that they could have added more statements to be printed in the terminal. For example, they could have written an explanation of the special food's score distribution when consumed by the snake, so players would understand.

2. **[3 marks]** Quickly summarize in point form the pros and cons of the C++ OOD approach in the project versus the C procedural design approach in PPA3.

   Pros:
   - Code can be reused.
   - Can quickly access code written earlier, and takes up less space too.
   - Inheritance.

   Cons:
   - More effort at first to create the object classes, especially compared to the C procedural design approach.
   - Could be harder to debug through multiple files.

Peer Code Review: Code Quality

1. **[3 marks]** Does the code offer sufficient comments, or deploys sufficient self-documenting coding style, to help you understand the code functionality more efficiently? If any shortcoming is observed, discuss how you would improve it.

This group added a lot of code in each of their cpp files. They also included a comment at the start of each cpp file that displayed the purpose of said file. I think having the purpose of the class commented in helps readers get a brief understanding of what each separate file does which is helpful. Also, their comments were strategically placed, so that it was enough to provide an understanding of the code, but not too many comments to overwhelm the reader.

2. **[3 marks]** Does the code follow good indentation, add sensible white spaces, and deploys newline formatting for better readability? If any shortcoming is observed, discuss how you would improve it.

   This code follows good indentation throughout the whole project and there was newline formatting placed. They also had a sensible amount of whitespace, where there was space between major code implementations. The only thing we noticed was that in the Project file, for example, there was whitespace inside if statements and for loops, but in the objPosArrayList cpp file, there was no whitespace between if statements and for loops. Both methods are perfectly readable, but for consistency throughout the code, I would suggest keeping the whitespace consistent throughout all their files in the project.

Peer Code Review: Quick Functional Evaluation

1. **[3 marks]** Does the Snake Game offer smooth, bug-free playing experience? Document any buggy features and use your COMPENG 2SH4 programming knowledge to propose the possible root cause and the potential debugging approaches you'd recommend the other team to deploy. (NOT a debugging report, just technical user feedback)

   There's no food generation initially until after the snake starts moving. The snake collision system seems to occur before the snake overlaps. That seems to be a purposeful design which makes sense however in terms of viewing it makes it difficult to see the collision as shown in Figure 1. As shown in the image, its hard to see what part of the snake is colliding. I think having an asterisk as the snake's body made it a little difficult to view the potential food candidates in terms of alignment since the asterisk isn't centered like the food items instead it's aligned to the top of the lines.

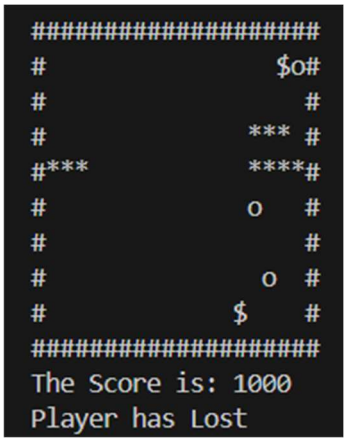

Figure 1: Snake Collision

2. **[3 marks]** Does the Snake Game cause memory leaks? If yes, provide a digest of the memory profiling report and identify the possible root cause(s) of the memory leakage.

   Dr. Memory was run and there was no memory leak shown, this can be confirmed when going through the code, because every new has a delete associated with it.

Project Reflection

Recall the unusual objPos class design with the additional Pos struct. After reviewing the other team's implementation in addition to your own, reflect on the following questions:

1. **[3 marks]** Do you think the compound object design of objPos class is sensible? Why or why not?

   The design of the objPos class is sensible, it follows the rule of 5 – default, specialized, copy constructors and copy assignment operator, as well as a destructor. Also, I found having the Pos struct in the objPos class was sensible because it allows for the variables inside the struct be accessible. Also, having the class makes it easy to make changes in the code that don't affect the rest of the code present in other files.

2. **[4 marks]** If yes, discuss an alternative objPos class design that you believe is relatively counterintuitive than the one in this project. If not, explain how you'd improve the object design. <u>You are expected to facilitate the discussion with UML diagram(s).</u>

   An alternative objPos class design would be to have the x and y integers be part of the objPos class without being within a struct. They could have been added as fields in objPos as they would be directly implemented into it. This way we would not have to access x and y through Pos first.

objPos
+ x: int + y: int +symbol: char
+ objPos() + objPos(xPos: int, yPos: int, sym: char) + setObj Pos (o: obj Pos): void + setObj Pos (xPos:int, yPos:int, sym: char): void + getObj Pos(): objPos const + getSymbol(): char const + isPosEqual (refPos: const objPos*): bool const + getSymbolIfPosEqual (refPos: const objPos*): char const